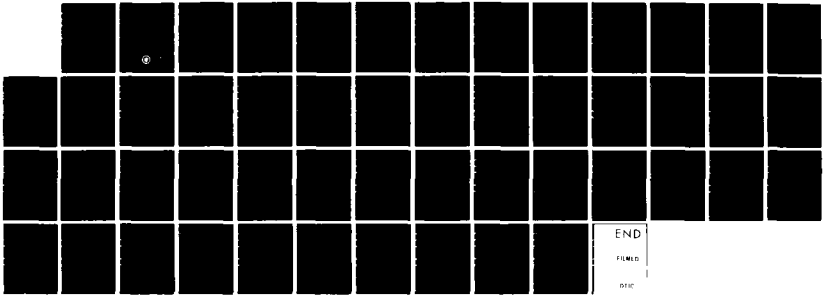
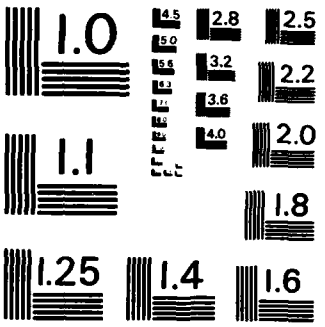
AD-A146 272 VERIFICATION OF A CLASS OF DEADLOCK-FREE SELF-TIMED COMPUTATIONAL ARRAYS(U) PITTSBURGH UNIV PA INST FOR COMPUTATIONAL MATHEMATICS AND APP.. R G MELHEM AUG 84 1/1

UNCLASSIFIED ICMA-84-76 N00014-80-C-0455 F/G 12/1 NL

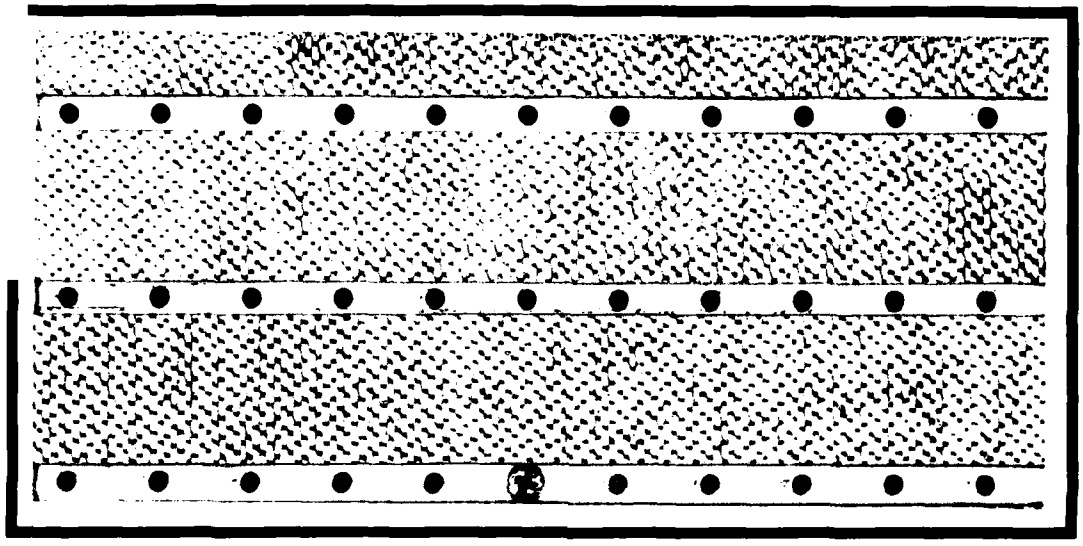




(15)

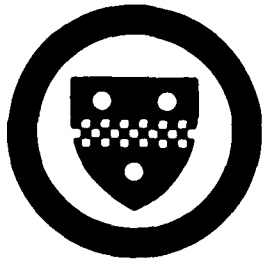
AD-A146 272

INSTITUTE FOR COMPUTATIONAL MATHEMATICS AND APPLICATIONS



Department of Mathematics and Statistics

University of Pittsburgh



Technical Report ICMA-84-76 August 1984

VERIFICATION OF A CLASS OF DEADLOCK-FREE, SELF-TIMED, COMPUTATIONAL ARRAYS*)

by

Rami G. Melhem**)

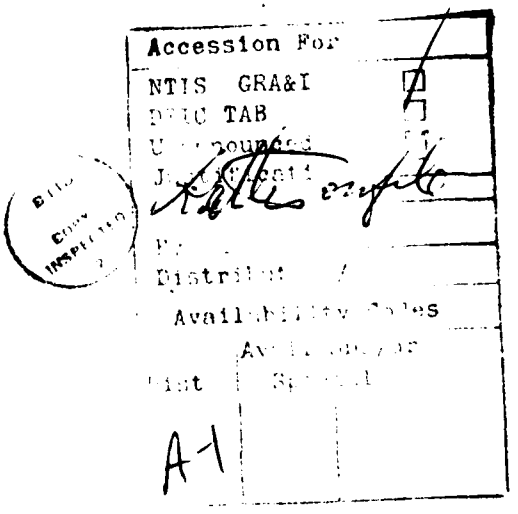
Institute for Computational Mathematics and Applications
Department of Mathematics and Statistics
University of Pittsburgh
Pittsburgh, PA 15260

*) This work in part was supported under ONR Contract N00014-80-C-0455.

**) The address of the author as of August 15, 1984 is: Dept. of Computer Science, Purdue University, West Lafayette, Indiana 47907.

ABSTRACT

A special class of self-timed networks is defined and two steps are suggested for the verification of any computation performed on networks in this class. First, a proof that the network is deadlock-free, and second, a verification of the results of the computation. In relation to the first step, an algebra of events is developed and used to prove that the liveness of any self-timed network is determined uniquely by its initial state. Moreover, a method is presented for the verification of liveness in networks preset to given initial states. In order to verify the results of deadlock-free self-timed computations, general VLSI cellular networks are defined to include both systolic and self-timed computational arrays. A model for the verification of computations on these networks is then obtained by the generalization of a previous model that was suggested for systolic computations.



1. INTRODUCTION

Systolic arrays [8] and self-timed systems [17] are two different approaches for the organization of large VLSI computing networks. In systolic arrays, a global clock is used for the synchronization of all the cells in the network. This global control leads often to simple designs, but may create problems in large arrays due to the non-simultaneous arrival of clock signals at the different cells (clock skew). On the other hand, synchronization in self-timed systems is achieved by local protocols between neighboring cells. This requires some additional hardward/firmware and, hence, may lead to expensive implementations.

The reliability of the operations of clocked networks depends on the ability to ensure the arrival of inputs and the collection of outputs on time. Namely, a failure to supply the network with an input item on time may result in its malfunction, and an output item that is not sampled on time may be overwritten by other data, and thus lost. If the network is managed by a host computer, then any mismatch between the nominal I/O rate of the network and the capability of the host to handle I/O may be detected during the design phase. However, problems may occur if the clocked network is a component in a larger system and receives its input from (or delivers its output to) other components that may have different I/O rates or, even worth, variable I/O rates.

A similar situation arises when a network is designed for use in real time environments, where the time of arrival of the input items is not determined a priori. Examples of such networks are the priority queue network [12] that may be used to hold service requests in an operating system, and the data base machine [1] where the time of arrival of a certain transaction is a real time event. These reasons, in addition to the ones argued

for in [10], may justify the consideration of the more complex and expensive self-timed computational arrays, despite the facts that equivalent clocked arrays may be designed for most self-timed arrays and that the effect of the clock skew may be kept under control [7].

Self-timed techniques (sometimes called data driven techniques) may be easy to apply on pipelined systems where data flows in one direction. However, it seems that little attention has been paid to the complications that may result from applying these techniques to networks with data flowing in more than one direction. More specifically, a network may be deadlocked if each cell is waiting to receive its input from other cells.

In this paper, we consider a class of self-timed computational networks where each cell does not start its computation before all its inputs are available, and does not output the results before all previous outputs are consumed. This class, simply called here self-timed networks, is precisely defined in Section 2 as a subclass of VLSI cellular networks; a more general class that includes also systolic networks.

The model that we suggest in Section 4 for the verification of the results of both clocked and self-timed computations is a generalization of a previous model [15] that was shown to be effective in the verification, as well as the simulation, of systolic computations. The basic idea is to represent the data items that appear on any communication link in the network by a data sequence, and to model the operation of each cell by a set of equations that includes operators on sequences. The verification of the computation may then be accomplished by solving the system of equations that models all the cells in the network, for specified inputs.

However, in order to keep the model relatively simple, we assume that the computation is deadlock-free, a temporal condition that is not satisfied

always in self-timed networks. Hence, a proof that a given self-timed computation is deadlock-free is necessary before the abstract model may be used to verify its results. The goal of Section 3 is to provide a technique for such proofs.

Many methods were introduced in the literature for the verification of temporal properties in communicating processes (see e.g. [2], [4]) and in asynchronous systems (see e.g. [16]). However, these techniques do not seem adequate for the type of results that we are seeking. More specifically, we want to prove that deadlock in self-timed networks depends uniquely on the initial state, and then identify the initial states that guarantee deadlock-free computations.

In order to obtain these results, we develop an algebra of events, where the execution history of a network is represented by a sequence of events and the state of the network, at any time, is represented by a state matrix. The occurence of each event causes a transformation in the state matrix according to some axiom. This algebra proved to be useful for the study of different properties of execution histories and their dependency on the initial state of the network. Similar algebras of events were introduced in [3] and [13] but used in different contexts.

Finally, we present an example in Section 5 to illustrate the ideas and techniques discussed in Sections 3 and 4.

2. VLSI CELLULAR NETWORKS

A VLSI cellular network is defined here as a set of cells, each having a certain number of input and/or output ports, and a set of unidirectional communication links, each connecting an output port of some cell to an input port of another cell.

A communication link ℓ_{ij} directed from an output port of a cell c_i to an input port of another cell c_j is a medium that carries always one data item of a certain type. At any instant, this medium may be in one of two states, namely RE (read enabled) or WE (write enabled). Both c_i and c_j may examine the state of ℓ_{ij}, but data may be written on it only by c_i and read from it only by c_j. The state of ℓ_{ij} is set to RE whenever a data item is written by c_i and to WE whenever its content is read by c_j. At the beginning of the operation of the network, ℓ_{ij} may be either initially read enabled (IRE) or initially write enabled (IWE). Note that the above abstract definition of a link is independent of any particular implementation of the medium or the states. We will use the usual convention of calling ℓ_{ij} an output link of c_i and an input link to c_j and we will assume that a link cannot be both an input and an output link to the same cell. This assumption does not affect the computational power of the network since any direct feedback to a cell may be included in its specification.

The cells in VLSI cellular networks may be classified into three classes; namely, 1) input cells that have only output ports, 2) output cells that have only input ports, and 3) computational cells that have both input and output ports. Each cell in the network repeats indefinitely the execution of a specific cycle that is synchronized by two flags; namely, STR (start to read) and STW (start to write). More specifically, each cell executes an algorithm of the following generic form.

Algorithm GENERIC

```
Repeat forever
Begin
1: If (STR = false) then go to 1          (* Busy wait *)
    else Begin
         reset STR := false;
         Read one data item from each input port (if any);
         end;

2:  Case (cell-type) of
    Computational:  Compute a data item for each output port using a specific
                    procedure applied to the data read since the beginning of
                    execution;
    Input       :   Create a data item for each output port;
    Output      :   Inhale the data items read in step 1;

3:  If (STW = false) then go to 3          (* Busy wait *)
    else Begin
         reset STW := false;
         Write on each output port (if any) the corresponding item computed or
         created in step 2;
         end;
end.
```

It is easy to see that systolic networks form a subclass of VLSI cellular networks in which all the links in a network are IRE and the flags STR and STW of all the cells are set simultaneously by a global two-phase clock. On the other hand, we may define self-timed networks as a subclass of VLSI cellular networks in which any link may be either IRE or IWE and each cell

sets its STR and STW flags asynchronously. Namely, a cell sets its STR flag when all its input links are RE and sets its STW flag when all its output links are WE. A cell with no input (or output) links sets the flag STR (or STW) as soon as it is reset.

In order to obtain a useful computational model, we assume that the execution of step 2 in algorithm GENERIC always terminates. This, in particular, assumes that a data item is eventually supplied whenever the network is waiting for an input and that the output of the network is eventually consumed. However, an output item does not have to be consumed as soon as it is produced. In fact, this is the reason for having an "Inhale" statement, that may require some time to execute, in step 2 of algorithm GENERIC.

The different steps in GENERIC provide a convenient way for describing the state of a certain cell during the execution of a network. More specifically, we may say that a cell is in state R (reading), C (computing) or W (writing) if it is executing step 1, 2 or 3 of GENERIC, respectively.

In Section 4, we will generalize the abstract systolic model of [15] to model VLSI cellular networks. This generalization, however, will be based on two assumptions, namely that

A1) Each cell in the network executes infinitely many cycles.

A2) A communication link is WE when a data item is written on it, and is RE when a data item is read from it.

The conditions needed to validate these assumptions are straightforward for systolic networks. Namely, A1 is valid if the global clock never stops and A2 is valid if the period of the clock and the shifts between its two phases are such that 1) the shift (ϕ_1) between phase 1 and phase 2 is large

enough to execute steps 1 and 2 of GENERIC in all the cells in the network, and 2) the phase shift between phase 2 and phase 1 (period - ϕ_1) is large enough to execute step 3 of GENERIC. Note that this should include the effect of the clock skew and the time needed for the signals to stabilize.

On the other hand, A2 is satisfied in self-timed networks by definition. However, A1 may be violated if the network reaches a state where each of its cells is trapped in a busy-wait loop in either step 1 or 3 of GENERIC. This is a typical deadlock situation that we will study in details.

3. A STUDY OF DEADLOCK IN SELF-TIMED NETWORKS

In the networks considered in this section, we assume that at most one link may be directed from a specific cell to another cell. We justify this assumption as follows: Let $\ell_{i,j}$ and $r_{i,j}$ be two links directed from cell i to cell j. If the initial states of $\ell_{i,j}$ and $r_{i,j}$ are different, then both cell i and cell j will be trapped in the busy waiting loop of step 3 and step 1, respectively, of algorithm GENERIC. Hence both cells will be blocked indefinitely and no further study is needed. If, on the other hand, the states of $\ell_{i,j}$ and $r_{i,j}$ are initially identical, then whenever cell i executes a write, or cell j executes a read, the states of both $\ell_{i,j}$ and $r_{i,j}$ will change. In this case, the states of the two links may be expressed by the state of a single link.

We will also assume that the network is connected in the sense that it cannot be partitioned into isolated subnetworks. If, however, such partitioning does exist, then deadlock in each subnetwork may be studied independently.

3.1 The State Matrix

Let N be a self-timed network composed of n cells labeled by the integers $1,\ldots,n$ and let the topology of N be specified by the adjacency matrix A defined such that $A_{i,j} = 1$ if there is a link $\ell_{i,j}$ directed from cell i to cell j, and $A_{i,j} = 0$ otherwise. At any instant during the execution of N, each link may be in one of two states, namely RE or WE. Hence, the states of all the links in N may be expressed in terms of an $n \times n$ matrix S defined as follows

$$S_{i,j} = \begin{cases} 0 & \text{if } A_{i,j} = 0 \\ 1 & \text{if } A_{i,j} = 1 \text{ and } \ell_{i,j} \text{ is RE} \\ -1 & \text{if } A_{i,j} = 1 \text{ and } \ell_{i,j} \text{ is WE} \end{cases} \tag{3.1}$$

Noting that we do not allow direct loops in N, it is clear that the diagonal elements of S are always zeroes. Hence, $S_{i,i}$, $i = 1,\ldots,n$, may be used to indicate the states of the n cells in N. More specifically, we set

$$S_{i,i} = \begin{cases} 1 & \text{if cell } i \text{ is in state } R \\ 0 & \text{if cell } i \text{ is in state } C \\ -1 & \text{if cell } i \text{ is in state } W \end{cases} \tag{3.2}$$

The matrix S defined by (3.1) and (3.2) is called the state matrix.

3.2 Events and Instances of Events

Given a particular cell i in a self-timed network, we refer to any change in the state of that cell by an "event". More specifically, the events $E_{r,i}$, $E_{c,i}$ and $E_{w,i}$ are defined as the events of leaving state R, leaving state C and leaving state W, respectively (see Figure 1). We also define the t-th instance $E_{x,i}^{(t)}$ of an event $E_{x,i}$, $x = r$, c or w, as the occurence of $E_{x,i}$ during the execution of the t-th cycle of cell i.

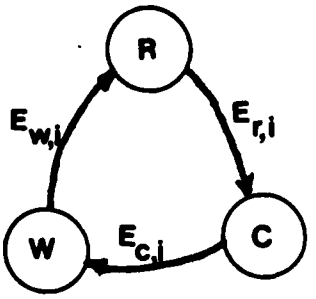


Figure 1: State transition diagram of cell i.

The statesof the links connected to cell i play a key role in the occurence of event_instances in that cell. For example, if cell i is executing step 1 in the t-th cycle of algorithm GENERIC, then it may not escape the busy waiting loop unless all its input links are RE. Once this condition is satisfied, the ELSE part of step 1 will start execution leading to the eventual termination of that step, that is to the occurence of $E_{r,i}^{(t)}$. In other words, the occurence of any instance of $E_{r,i}$ is feasible only if cell i is in state R and all its input links are RE. Similarly, it may be seen that the occurence of any instance of $E_{w,i}$ is feasible only if cell i is in state W and all its output links are WE. Finally, the occurence of any instance of $E_{c,i}$ is feasible only if cell i is in state C. This argument motivates the following definitions:

Definition 1: Let A be the adjacency matrix of a network N composed of n cells. Then the domain of any event $E_{x,i}$, $1 \leq i \leq n$, is defined as follows:

$$\text{dom}(E_{x,i}) = \begin{cases} \{(j,i) \mid 1 \leq j \leq n,\ A_{j,i} = 1\} \cup \{(i,i)\} & \text{if } x = r \\ \{(i,i)\} & \text{if } x = c \\ \{(i,j) \mid 1 \leq j \leq n,\ A_{i,j} = 1\} \cup \{(i,i)\} & \text{if } x = w \end{cases}$$

Definition 2: Given a state S of a network N, an event $E_{x,i}$ is said to be feasible at S if

For $x = r$	$S_{j,i} = 1$	$(j,i) \in \text{dom}(E_{x,i})$
For $x = c$	$S_{i,i} = 0$	$(i,i) \in \text{dom}(E_{x,i})$
For $x = w$	$S_{i,j} = -1$	$(i,j) \in \text{dom}(E_{x,i})$

Definition 3: A state S is called a dead state if no events are feasible at S.

Clearly, the occurence of an event instance changes the state of the network. More specifically, for any positive integer t, the occurence of $E_{r,i}^{(t)}$ changes the state of cell i from R to C. Also as a result of reading the inputs, the states of the input links to cell i are set to WE. Similarly, the occurence of $E_{w,i}^{(t)}$ changes the state of cell i from W to R, and as a result of writing, the states of the output links of cell i are set to RE. Finally, the occurence of $E_{c,i}^{(t)}$ changes the state of cell i from c to w. This is formally described by the following axiom.

State Transformation Axiom

Let $E_{x,i}$ be an event feasible at a state S of a network N, and let $\bar{S}$ be the state of N after the occurence of an instance $E_{x,i}^{(t)}$ of $E_{x,i}$. If $E_{x,i}^{(t)}$ is the only instance that occurred between S and $\bar{S}$, then

1) For $(i,j) \notin dom(E_{x,i})$ $\quad \bar{S}_{i,j} = S_{i,j}$

2) For $(i,j) \in dom(E_{x,i})$

If $x = r$ then $\quad \bar{S}_{i,j} = \begin{cases} -1 & i \neq j \\ 0 & i = j \end{cases}$

If $x = c$ then $\quad \bar{S}_{i,j} = -1$

If $x = w$ then $\quad \bar{S}_{i,j} = 1$

In the following section we consider the interaction between the different

cells in the network.

3.3 The Execution History of Self-Timed Networks

An execution history of a self-timed network N may be expressed by a sequence $V_1, V_2, \ldots$, where each V_i is a set that contains event instances. Every event instance that occured during execution should appear in some V_i such that 1) each set contains instances that occured simultaneously, and 2) V_{i-1} contains instances that occured before those in V_i. Note that if N does not deadlock, then $V_1, V_2, \ldots$ is infinite.

Let S be some state of N. We will use the notation $S \xrightarrow{V} \bar{S}$, where V is a set of event instances, to indicate that each event in V is feasible at S and that $\bar{S}$ is the state of N after the simultaneous occurence of the instances in V.

Given an initial state S^0 for N, it should be clear that different runs of N may result in different execution histories, depending on the time that each cell spends in executing the different steps of its cycles. However, any particular execution history, say $V_1, V_2, \ldots$, should satisfy the following properties:

[P1] There exist a sequence of states $S^0, S^1, \ldots$, associated with $V_1, V_2, \ldots$ such that, for any $i > 0$, $S^{i-1} \xrightarrow{V_i} S^i$. That is, at any instant during execution, only instances of feasible events may occur.

[P2] An instance $E^{(t)}_{x,k}$, for some $t \geq 1$, $1 \leq k \leq n$ and $x = r$, c or w, may occur at most once in $V_1, V_2, \ldots$. Moreover, if $t > 1$ and $E^{(t)}_{x,k} \in V_i$ for some i, then there exist a $j < i$ such that $E^{(t-1)}_{x,k} \in V_j$. That is, the (t-1)-st instance of an event should occur before its t-th instance.

P3 If $V_1, V_2, \ldots, V_m$ is finite, then S^m is a dead state. That is, execution stops only if N is deadlocked.

Definition 4: Any sequence of sets of event instances that satisfies the above three properties is called an S^o-feasible sequence.

By definition, any execution history of N starting from S^o is an S^o-feasible sequence. The mathematical definition of feasible sequences makes them easy to construct and study. The following lemmas and definitions are needed in order to prove some properties about feasible sequences, and thus execution histories.

Lemma 1: If $E_{x,k}$ and $E_{y,q}$ are two different events feasible at a certain state S of N, then

$$\mathrm{dom}(E_{x,k}) \cap \mathrm{dom}(E_{y,q}) = \phi. \tag{3.3}$$

Proof: If $x \neq y$, then (3.3) follows directly from Definition 2. If $x = y$, then from Definition 1, (3.3) is false only if $k = q$, which contradicts the hypothesis that $E_{x,k}$ and $E_{y,q}$ are different. ■

From the above lemma, it is clear that if $S^a \xrightarrow{V} S^{a+1}$, where $V = \{e^{(t)}, v^{(t)}\}$, then $\mathrm{dom}(e) \cap \mathrm{dom}(v) = \phi$ and hence S^{a+1} may be obtained from S^a by applying the state transformation axiom to e and v independently.

The following lemma proves that if an event is feasible at a certain state then it remains feasible until the occurence of one of its instances.

Lemma 2: Let e be an event feasible at S^a and let $S^a \xrightarrow{V_{a+1}} S^{a+1} \longrightarrow \ldots \xrightarrow{V_m} S^m$. If $V_{a+1},\ldots,V_m$ do not contain any instance of e, then e is feasible at S^m.

Proof: Let $v^{(t)} \in V_{a+1}$. Hence, v is feasible at S^a and so is e (from the hypothesis). Then, by Lemma 1, $dom(e) \cap dom(v) = \phi$. But from the state transformation axiom, the occurence of $v^{(t)}$ does affect only the elements of S^a in $dom(v)$. Noting that this is valid for any $v^{(t)}$ in V_{a+1}, we conclude that $S^{a+1}_{i,j} = S^a_{i,j}$ for $(i,j) \in dom(e)$ and hence e is feasible at S^{a+1}. The feasibility of e at S^m follows by repeating the above argument $m-a$ times. ■

Next, we redefine Lamport's partial ordering of events [11] to suit our particular model.

Definition 5: Two even instances I_1 and I_2 are related by the ordering $I_1 <_{S^o} I_2$ if for any S^o-feasible sequence $V_1,V_2,\ldots$, $I_2 \in V_i$, for some i, implies that $I_1 \in V_j$, for some $j < i$. That is if I_2 does occur then I_1 should occur before I_2.

By definition, the relation $<_{S^o}$ is associative; that is, if $I_1 <_{S^o} I_2$ and $I_2 <_{S^o} I_3$, then $I_1 <_{S^o} I_3$. Note that $I_1 <_{S^o} I_2$ does not imply that $I_2 \in V_i$ for some i. The subscript S^o will be dropped from $<_{S^o}$ when it is obvious from the context.

Lemma 3: Given an initial state S^o for N, the following partial orderings do hold for any S^o-feasible sequence.

I) For any i and $t \geq 1$,

$$E^{(t)}_{r,i} < E^{(t)}_{c,i} < E^{(t)}_{w,i} < E^{(t+1)}_{r,i}. \tag{3.4a}$$

II) For $t \geq 1$ and any i and j with $S^o_{i,j} = 1$,

$$E^{(t)}_{r,j} < E^{(t)}_{w,i} < E^{(t+1)}_{r,j}. \tag{3.4b}$$

III) For $t \geq 1$ and any i and j with $S^o_{i,j} = -1$,

$$E^{(t)}_{w,i} < E^{(t)}_{r,j} < E^{(t+1)}_{w,i}. \tag{3.4c}$$

Proof: We will prove here only one of the seven inequalities involved in the lemma. The proof of the other six is very similar. More specifically we will prove, by induction on t, that if $S^o_{i,j} = 1$ then $E^{(t)}_{r,j} < E^{(t)}_{w,i}$. Let $V_1, V_2, \ldots$ be S^o-feasible such that $S^o \xrightarrow{V_1} S^1 \xrightarrow{V_2} \ldots$.

for $t = 1$: Assume that $E^{(1)}_{w,i} \in V_z$ for some z, then $E_{w,i}$ is feasible at S^{z-1} and hence $S^{z-1}_{i,j} = -1$. But $S^o_{i,j} = 1$ and, by the state transformation axiom, the (i,j)-th entry of the state matrix may be set to -1 only by an instance of $E_{r,j}$. Then there exist T and v such that $v < z$ and $E^{(T)}_{r,j} \in V_v$. If $T > 1$ then, by property P2 of feasible sequences, there exist a $q < v < z$ such that $E^{(1)}_{r,j} \in V_q$. This is valid for any S^o-feasible sequence and hence, $E^{(1)}_{r,j} < E^{(1)}_{w,i}$.

for $t > 1$: Assume that $E^{(t-1)}_{r,j} < E^{(t-1)}_{w,i}$. If $E^{(t)}_{w,i} \in V_z$, for some z, then, from property P2, there exists a $u < z$ such that

$E_{w,i}^{(t-1)} \in V_u$. But the feasilbility of $E_{w,i}^{(t)}$ at S^{z-1} implies that $S_{i,j}^{z-1} = -1$ and from the state transformation axiom, $S_{i,j}^{u} = 1$, and only an instance of $E_{r,j}$ may set the (i,j)-th entry of the state matrix to -1. Then, there exist T and v such that $u < v < z$ and $E_{r,j}^{(T)} \in V_v$. However, by the induction hypothesis $E_{r,j}^{(t-1)}$ occured before $E_{w,i}^{(t-1)}$ and hence, $T \geq t$. Finally, if $T > t$, then there exist a $q < v < z$ such that $E_{r,j}^{(t)} \in V_q$. This is valid for any S^0-feasible sequence and hence, $E_{r,j}^{(t)} < E_{w,i}^{(t)}$. ■

Lemma 4: If $dom(e) \cap dom(v) \neq \Phi$, then for any initial state S^0 and any $t > 1$, there exist a $T \geq 1$ such that

$$e^{(T)} <_{S^0} v^{(t)}. \tag{3.5}$$

Proof: From Definition 1, it is clear that the following are the only cases where the domains of two events do intersect.

1) $dom(E_{r,j}) \cap dom(E_{w,i}) = \{(i,j)\}, \quad i \neq j$

2) $dom(E_{x,i}) \cap dom(E_{y,i}) = \{(i,i)\}, \quad x,y = r, c \text{ or } w, \quad x \neq y.$

The proof of (3.5) in the first case follows from (3.4b/c) and in the second case follows from (3.4a). ■

Now, we are ready to prove that the initial state is the only factor that affects the deadlock of a self-timed network, irrespective or any particular execution history.

Theorem 1: Given an initial state S^0 of N, if there exists a finite S^0-feasible sequence, then any other S^0-feasible sequence is finite. Moreover, if there exists an infinite S^0-feasible sequence, then any other S^0-feasible sequence is infinite.

Proof: See the appendix.

Hence, for some initial states, every execution of N will end in a deadlock, while for other initial states, N is deadlock-free. This is made more precise by the following definition.

Definition 6: An initial state S^0 is live if there exists an infinite S^0-feasible sequence. Otherwise S^0 is non-live.

By Theorem 1, any execution of a network N starting at a live initial state may not deadlock. On the other hand, if N starts execution from a non-live initial state, then it will definitely reach a dead state.

3.4 Determination of Live Initial States

Theorem 1 allows the use of any especially constructed feasible sequence for testing the liveness of initial states. In this section, we will consider sequences where events in the first cycle are always given a higher priority to occur. More specifically we define "First Cycle First" (FCF) event sequences as follows:

Definition 7: An S^0-FCF sequence is a sequence of sets of event instances, $V_1, V_2, \ldots,$ constructed as follows:

1) $V_1 = E^{(1)}_{x,k}$ where $E_{x,k}$ is any event feasible at S^0.

2) For any $i \geq 1$, let $S^0 \xrightarrow{V_1} S^1 \longrightarrow \ldots \xrightarrow{V_i} S^i$ and let F_{i+1} be the set of events feasible at S^i. If $F_{i+1} = \phi$ then terminate the construction, else if there exist an event $E_{y,q} \in F_{i+1}$ such that $E^{(1)}_{y,q}$ did not occur in $V_1,\ldots,V_i$, then take $V_{i+1} = \{E^{(1)}_{y,q}\}$. Otherwise, set $V_{i+1} = \{E^{(t)}_{z,p}\}$ for any $E_{z,p} \in F_{i+1}$, where t is such that $E^{(t-1)}_{z,p}$, but not $E^{(t)}_{z,p}$, did occur in $V_1,\ldots,V_i$. ■

S^0-FCF sequences are, by construction, S^0-feasible sequences and hence they may be used to determine the liveness of S^0. More specifically we may prove the following.

Theorem 2: Let $V_1,V_2,\ldots$ be any S^0-FCF sequence and let m be the smallest integer such that $V_m = \{E^{(t)}_{x,k}\}$ for some x,k and $t \neq 1$. Then S^0 is a live initial state if and only if $m = 3n$, where n is the order of the matrix S^0, which is the number of cells in the corresponding network.

Proof: See the appendix.

By applying Theorem 2, we may test the liveness of a given initial state S^0 of a network N by simulating an execution history of N where only the first cycle (3 events) is allowed to execute in any cell. If in this execution every cell is able to complete its first cycle then S^0 is live, otherewise it is not. The following algorithm, LIVE, does exactly that. More specifically it simulates the execution of the first m instances of an S^0-FCF sequence. At any particular state, feasible events are identified and instances of these

events are executed causing the appropriate state transformations. However, only the first instance of any event is allowed to occur by marking any cell that completes its first cycle and disabling any event in marked cells. The simulation terminates when no events are feasible in unmarked cells. The initial state is then declared live if and only if, all the cells are marked. Note that for any i, $E_{c,i}$ becomes feasible immediately after the occurence of $E^{(1)}_{r,i}$. Hence, $E^{(1)}_{r,i}$ and $E^{(1)}_{c,i}$ may be executed in a single step as shown in the following algorithm

Algorithm LIVE

```
INPUT:  S[ ]:= S^0;
FOR  i := 1  TO  n  DO     Mark[i]:= False;
REPEAT
     Blocked := True
     FOR  j := 1  TO  n  DO
          IF (NOT Mark[j]) AND (S[i,j] ≠ -1,  i = 1,...,n) THEN
          BEGIN                (* E_{r,j}  is feasible.  Execute  E^(1)_{r,j}  and  E^(1)_{c,j} *)
            FOR  i := 1  TO  n  DO   IF (S[i,j] = 1) THEN  S[i,j] := -1;
            Blocked := False;
          END;
     FOR  i := 1  TO  n  DO
          IF (NOT Mark[i]) AND (S[i,j] ≠ 1,  j = 1,...,n) THEN
          BEGIN                    (* E_{w,i}  is feasible.  Execute  E^(1)_{w,i} *)
               FOR  j := 1  TO  n  DO    IF(S[i,j] = -1) THEN  S[i,j] = 1;
               Blocked := False;
               Mark[i] := True;
          END;
```

UNTIL (Blocked);

IF (Mark[i] = True, i = 1,...,n) THEN S^o is live

ELSE S^o is not live

Next, we would like to determine the liveness of an initial state by a static analysis, that is without any simulation. This analysis may be based on the properties of the matrices corresponding to live initial states. For example, it is easy to show that if the matrix S^o does not contain any -1, then algorithm LIVE will terminate successfully in one iteration. Also if S^o is such that $S^o_{i,j} \neq 1$ if $i < j$ and $S^o_{i,j} \neq -1$ if $i > j$, then LIVE will terminate successfully in at most n iterations. However, the structure of the state matrix does depend on the particular numbers assigned to the cells of the network, and hence all possible numbering schemes should be considered. In fact it may be proven that an initial state is live if and only if there exist a numbering of the cells of the network such that the corresponding matrix S^o satisfies

$$\text{IF } S^o_{i,j} = -1 \text{ THEN } i < j \text{ and } S^o_{i,q} \neq 1 \text{ for } q = j,\ldots,n.$$

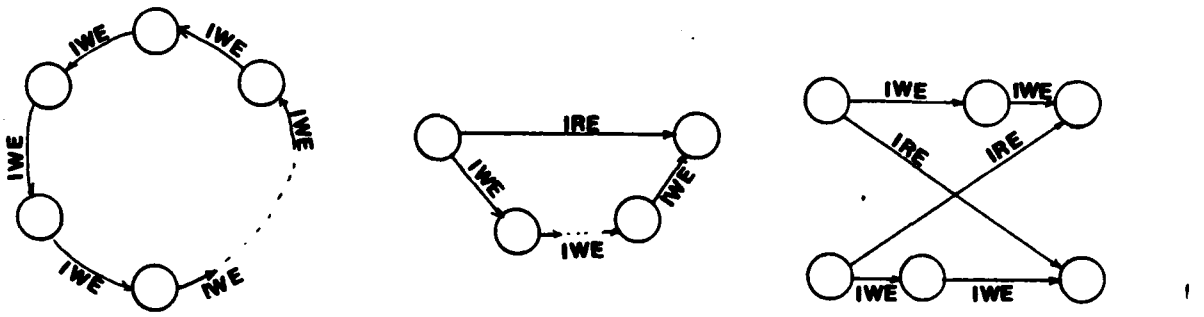
That is, a -1 may only appear in a upper diagonal and should not be followed by a 1 in the same row.

A liveness condition that is independent of any particular numbering of the cells, may only be obtained by an analysis of the graph underlying the initial state, rather than its matrix representation. We pursue this approach by first defining the initial state graph as follows:

Definition 8: The initial state graph of a network N is a labeled directed

graph $G_0 = (V,E,L)$, where V is a set of nodes corresponding to the cells in N, E is a set of edges corresponding to the links in N and $L: E \rightarrow \{IRE,IWE\}$ is a function that assigns a label from $\{IRE,IWE\}$ to each edge in E.

Definition 9: If e and v are two nodes in V, then we say that v is w-reachable from e, denoted by $e \rightsquigarrow v$, if there exist in G_0 a directed path P from e to v such that any edge in P is IWE. We also say that v is an R-neighbor of e, denoted by $e \rightarrow v$, if there exist an IRE edge directed from e to v.



(a) (b) (c)

Figure 2 - Initial configurations leading to deadlock.

Particular configurations in the initial state graph may be easily shown to cause deadlock. Consider, for example, the following cases:

Case 1: If there exist a loop with all its edges IWE (see Figure 2a). In this case every cell in the loop will not be able to complete its first read.

Case 2: If there exist an IRE edge that bypasses one or more IWE edges (see Figure 2b). In this case neither the source cell of the IRE link will be able to write nor its destination cell will able to read.

Case 3: If there exist two IRE cross links between two IWE paths (see Figure 2c). In this case neither the source cells of the IRE links will be able to write nor their destination cells will be able to read.

Clearly, the existance of any of the above cases in a network is sufficient to lead the network to a deadlock. However, is the absence of these cases sufficient to ensure that the network may not deadlock? The positive answer to this question is given by the following theorem.

Theorem 3: An initial state of a self-timed network N is live if and only if its initial state graph $G_0 = (V,E,L)$ satisfies all of the following conditions:

C1) *there does not exist a node* $e \in V$ *with* $e \rightsquigarrow e$,

C2) there does not exist two nodes $e,v \in V$ with $v \rightsquigarrow e$ and $v \rightarrow e$,

C3) there does not exist 2k nodes $e_i, v_i \in V$, $i = 0,\ldots,k-1$, for any $k \geq 1$ with $v_i \rightsquigarrow e_i$, $i = 0,\ldots,k-1$, $v_{i-1} \rightarrow e_i$, $i = 1,\ldots,k-1$ and $v_{k-1} \rightarrow e_0$.

Proof: See the appendix.

The first two conditions are formal expressions for cases 1 and 2 discussed above. The third condition is a general form of case 3, where k IRE cross links between k IWE paths form a cycle. It should be mentioned that the verification of these conditions may be straightforward in regular graphs corresponding to regular computational arrays. However, it is expected

that any algorithm for checking these conditions in a general graph will be of time complexity at least equal to $O(n^3)$, which is the complexity of the algorithm LIVE.

Having discussed different methods for the verification of the liveness of a self-timed computation, we present in the next section a model that may be used for the verification of its results.

4. A MODEL FOR DEADLOCK-FREE, VLSI CELLULAR NETWORKS

The systolic model introduced in [15] may be easily generalized to model computations performed on VLSI cellular networks. However, in order to keep the general model as simple as possible, we assume that every cell in the network executes infinitely many cycles. This, of course, excludes from the model any self-timed computation that starts execution from a non-live state. But, as proved in Section 3, this type of computations reaches a dead state before all its cells complete the first cycle, that is before any useful result is produced. Hence only trivial computations are excluded from the model by this assumption.

We also assume that the cycles of the different cells in the network are synchronized such that the state of any link ℓ_{ij}, directed from a cell i to a cell j, is WE when cell i writes a data item on ℓ_{ij} and is RE when cell j reads a data item form ℓ_{ij}. This assumption was validated in Section 2 for both systolic and self-timed synchronizations. It is needed to ensure that data on ℓ_{ij} are not lost or read more than once. More specifically, given the definition of the RE and WE states, the state transition diagram of ℓ_{ij} may be represented as in Figure 3. However, assuming that $E_{r,j}$ and $E_{w,i}$ do occur only when ℓ_{ij} is RE and WE, respectively, the two reflexive transitions should be removed, and hence, instances of $E_{r,j}$ and $E_{w,i}$ should occur alternatively. In other words, any data that is initially on ℓ_{ij} or written on it by cell i is read exactly once by cell j.

Before presenting the abstract model, we review briefly the definitions of data sequences and sequence operators of [15].

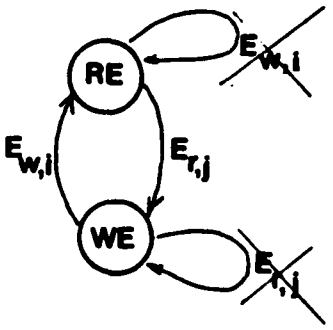


Figure 3: The state transition diagram of ℓ_{ij}

4.1 Data Sequences and Sequence Operators

Let N^+ be the set of positive integers, R be a set of data items of a certain type (for example, real numbers, boolean,...), and set $R_\delta = R \cup \{\delta\}$, where δ is a special element called the "don't care". A data sequence η is then defined as an infinite sequence of elements in R_δ; that is, a function $\eta: N^+ \to R_\delta$. Let $\bar{R}_\delta$ be the set of all data sequences that contain at most a finite number of non-δ elements.

Sequence operators are operators on $\bar{R}_\delta$. One way of defining sequence operators is to extend known operators from R to $\bar{R}_\delta$ by applying the operators element-wise to the elements of sequences. For example, if R is the set of real numbers, then the sequence addition $+: \bar{R}_\delta \times \bar{R}_\delta \to \bar{R}_\delta$ may be defined as follows:

$$\eta = \xi + \zeta \Rightarrow \eta(t) = \begin{cases} \delta & \text{if } \xi(t) \text{ or } \zeta(t) = \delta \\ \xi(t) + \zeta(t) & \text{if } \xi(t), \zeta(t) \neq \delta \end{cases}$$

Sequence operators may also be defined directly on $\bar{R}_\delta$, for example, we may define for any integer $r > 0$ the shift operator $\Omega_x^r: \bar{R}_\delta \to \bar{R}_\delta$, $x \in R_\delta$; as follows:

$$\eta = \Omega_x^r \xi \Rightarrow \eta(t) = \begin{cases} x & \text{if } t \leq r \\ \xi(t-r) & \text{if } t > r \end{cases}$$

That is, Ω_x^r inserts r elements equal to x at the beginning of its operand. If $x = \delta$ then Ω_x^r is simply written as Ω^r. Other sequence operators are defined in [14] and [15].

An operator $\Gamma: [\bar{R}_\delta]^n \to \bar{R}_\delta$ is called a causal operator if the t-th element of its image does not depend on any element $\sigma \geq t$ in any of its operands. That is, if $\eta = \Gamma(\xi_1,\ldots,\xi_n)$ then $\eta(t)$ does not depend on any $\xi_i(\sigma)$, $i = 1,\ldots,n$, $\sigma \geq t$. If the condition $\sigma \geq t$ is replaced by $\sigma > t$, then Γ is called a weakly causal operator. Causal and weakly causal operators play a key role in the model presented in the next section.

4.2 *The General Model*

Given a specific VLSI cellular network, we let R be the set of data items that may be transmitted on any communication link in the network, and we assume that each cell executes infinitely many cycles. Hence, with each input port (or output port) of a cell, we may associate an infinite data sequence on R_δ. This sequence represents the data items that are read (or written) during the execution of consecutive cycles. In other words, we may associate two data sequences with any communication link $\ell_{i,j}$. Namely, the sequence of data written on $\ell_{i,j}$ by cell i, say $\lambda_{i,j}$, and the sequence of data read from $\ell_{i,j}$ by cell j, say $\bar{\lambda}_{i,j}$. Assuming that no data is lost or read twice, then $\lambda_{i,j}$ and $\bar{\lambda}_{i,j}$ may be related by

$$\lambda_{i,j} = \begin{cases} \bar{\lambda}_{i,j} & \text{if } \ell_{i,j} \text{ is IWE} \\ \Omega_x \bar{\lambda}_{i,j} & \text{if } \ell_{i,j} \text{ is IRE} \end{cases} \qquad (4.1)$$

where x is the initial data on $\ell_{i,j}$.

The relation between the sequences on the output ports and those on the input ports of a computational cell may be determined from the specific details of step 2 of the algorithm GENERIC executed by the cell. More specifically, let $\gamma_1,\ldots,\gamma_m$ be the sequences of data read from the m input links of a particular computation cell i, and let $\bar{\lambda}_{i,j}$ be the sequence of data items written by cell i on any particular output link $\ell_{i,j}$. Then for any t, the data item $\bar{\lambda}_{i,j}(t)$ is computed during the t-th cycle of cell i using the data read since the beginning of execution; that is, using any $\gamma_i(\sigma)$, $1 \leq i \leq m$, $\sigma \leq t$. In other words, there exists a weakly causal operator $\Gamma: [\bar{R}_\delta]^m \to \bar{R}_\delta$ such that

$$\bar{\lambda}_{i,j} = \Gamma(\gamma_1,\ldots,\gamma_m). \tag{4.2}$$

The sequences of data read from and written on any given link are related by (4.1). Hence, only one sequence needs to be specified for each link. In our model, this is chosen to be the sequence of data read from the link. Any equation that relates the data on different links should then be written in terms of the sequences of data read from these links. For example, equation (4.2) should be rewritten using (4.1) in the form

$$\lambda_{i,j} = \begin{cases} \Gamma(\gamma_1,\ldots,\gamma_m) & \text{if } \ell_{i,j} \text{ is IWE} \\ \tilde{\Gamma}(\gamma_1,\ldots,\gamma_m) & \text{if } \ell_{i,j} \text{ is IRE} \end{cases} \tag{4.2}$$

where $\tilde{\Gamma} = \Omega_x \Gamma$ is a causal operator.

With this introduction, we may now define the abstract model for VLSI cellular networks to be composed of the following components:

[A1] A directed multigraph $G = (V,E)$ that describes the topology of the network, where V is a set of nodes and E is a set of edges, with no edge directed from a node to itself. Nodes and edges in G correspond to cells and links, respectively, in the network. Each node is identified by a unique label, and each edge is identified by a pair (y,v), where v is the label of the node at which it terminates and y is a color assigned to that edge. Edge colors do not have to be distinct. However, for unique identification of edges, different edges terminating at the same node should have different colors, and different edges starting at the same node should also have different colors. We will often refer to (y,v) as y_v.

[A2] For each edge in E a sequence is specified. We will denote the sequence associated with a link y_v by η_v, where η is the greek letter corresponding to y.

[A3] For each node v with q output edges ($q > 0$) and m input edges ($m > 0$), we are given q, m-ary weakly causal operators $\Gamma_v^i: [\bar{R}_\delta]^m \to \bar{R}_\delta$ $i = 1,\ldots,q$. These operators model the operation of the cell corresponding to v. More specifically, if $\zeta^1,\ldots,\zeta^q$ and $\eta_v^1,\ldots,\eta_v^m$ are the sequences associated with the output and input edges of v, respectively, then the q relations

$$\zeta^i = \Gamma_v^i(\eta_v^1,\ldots,\eta_v^m), \quad i = 1,\ldots,q$$

describe the operation of the cell. Here different edges are distinguished by their colors.

Clearly, the new interpretation of data sequences is more general than the

one given in [15]. However, the only difference in the components of the two models is the restriction imposed on the operators Γ_v^i. Namely, Γ_v^i are required to be causal in [15] while here we allow also weakly causal operators. This weaker restriction includes IWE links in the model in addition to the IRE links assumed in systolic networks. Note that the model does not explicitly distinguish between IRE and IWE links. This distinction is implicit in the type of operator used to describe the data on a particular link (see (4.3)).

Hence, the same techniques illustrated in [15] may be applied to the verification of self-timed networks as well as systolic networks. Namely, the system of sequence equations describing the operation of all the computational cells in the network is written, and then solved to obtain the sequences on the output links of the network (the input links to output cells) in terms of the sequences on its input links (the output links of input cells). This is illustrated in the next section by an example.

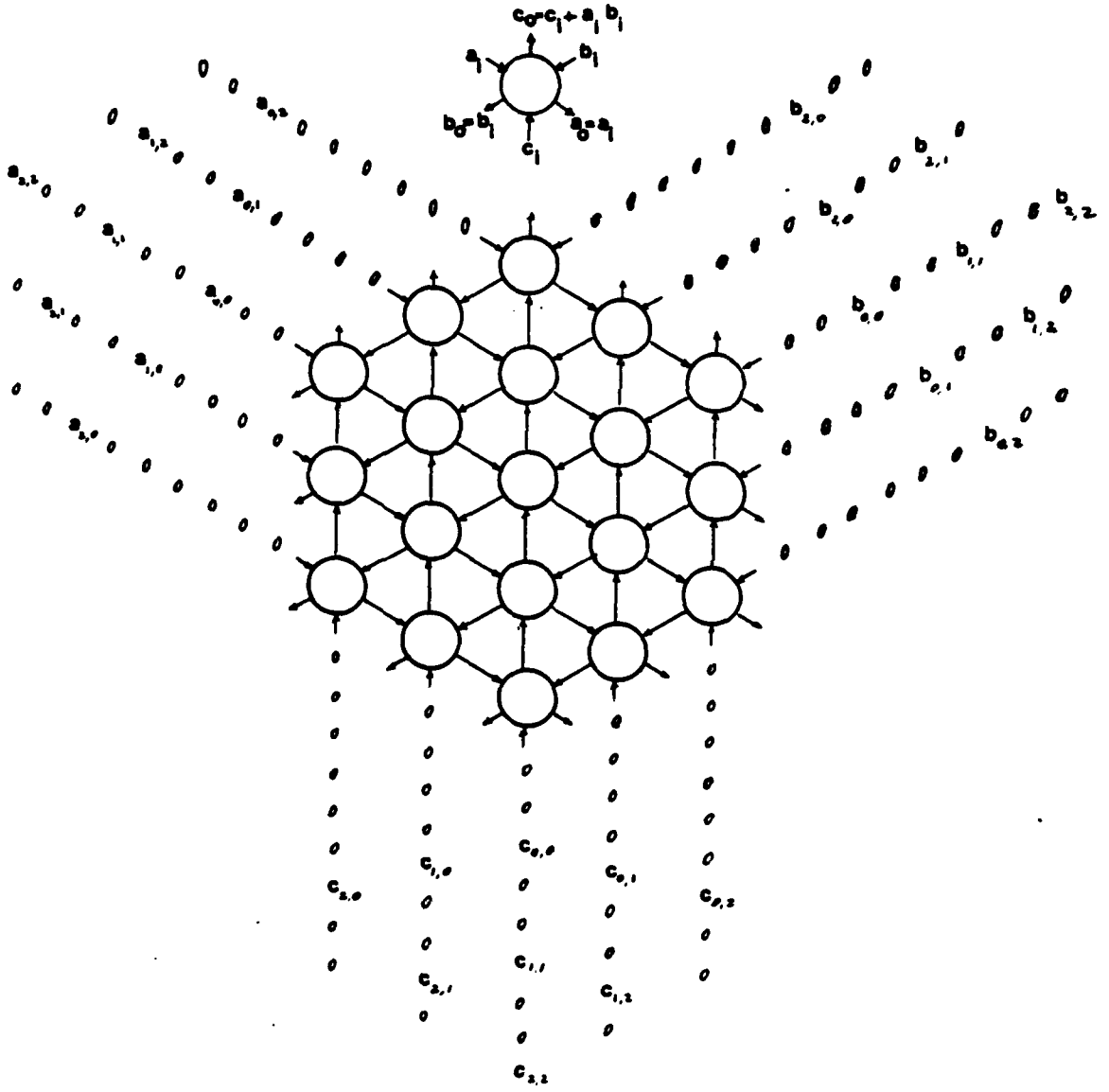
5. AN EXAMPLE: A FULL MATRIX MULTIPLICATION NETWORK

A systolic network for the multiplication of two full, square matrices, A and B, of order (n+1) was presented in [5]. The interconnection between the computational cells in this network for n = 2 is shown in Figure 4, where we also use the traditional pictorial representation to specify the input required for the proper operation of the network. As in the case of any systolic network, *every* link in the network is assumed to be IRE, carrying a don't care δ.

A self-timed version of the network may be directly derived from the systolic one, by changing only the synchronization mechanism. That is removing the global clock and allowing each cell to determine the beginning and the end of its cycles as described in Section 2. All the links in the network are kept IRE with δ on them and the input streams remain as shown in Figure 4. The resulting computation may be shown, by Theorem 3, to be deadlock-free.

In fact, a self-timed version of any systolic computation may be derived by changing only the synchronization mechanism. This type of self-timed systems is guaranteed to be deadlock-free provided that δ is treated as a data item that is read and written, rather than a "nothing". However, this is not usually the intention in self-timed networks. For instance, in the wave front machine introduced in [9], and in the self-timed versions of the networks suggested in [18], all the links in the network are assumed to carry "nothing", initially. That is, in our terms, all links are IWE.

By Theorem 3, a self-timed network with only IWE links is deadlock-free only if it does not contain any feedback. However, if the network contains a feedback, then, again by Theorem 3, any computation on this network will end in a dead state. For example, if in the network of Figure 4 all the links are initially set to the WE state, then every cell will be blocked waiting for



$$\begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} \\ c_{1,0} & c_{1,1} & c_{1,2} \\ c_{2,0} & c_{2,1} & c_{2,2} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{bmatrix}$$

Figure 4 - A systolic network for full-matrix multiplication.

some input data from a neighboring cell.

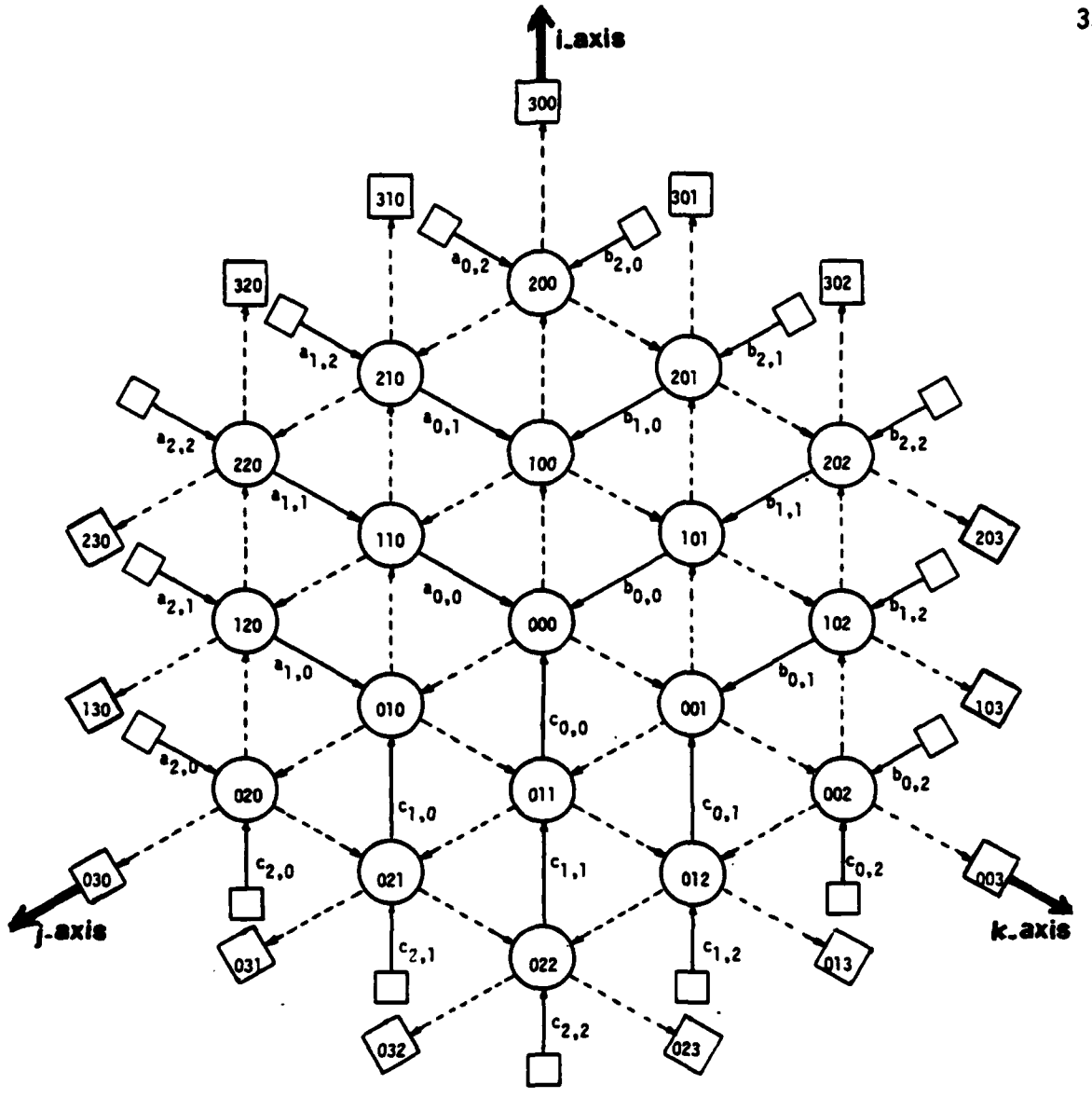
A self-timed version of the matrix multiplication network was presented and verified in [5] and [6]. The case $n = 2$ is shown in Figure 5 where IWE links are represented by dotted lines and IRE links by solid lines. The initial data on IRE links are also shown. Note that all the elements of the matrices A and B are used as initial data, and that the zeroes that was required to pad these elements in systolic operation are removed.

It is easy to check that the initial state of this network does satisfy the conditions of Theorem 3. Hence, the computation is deadlock-free. However, how does this self-timed network compare with the one derived from the systolic network of Figure 4 by simply changing the synchronization mechanism?. More specifically, will it produce the results at a faster rate? The answer in this particular case is negative, but more research is needed in order to find the answer for general networks.

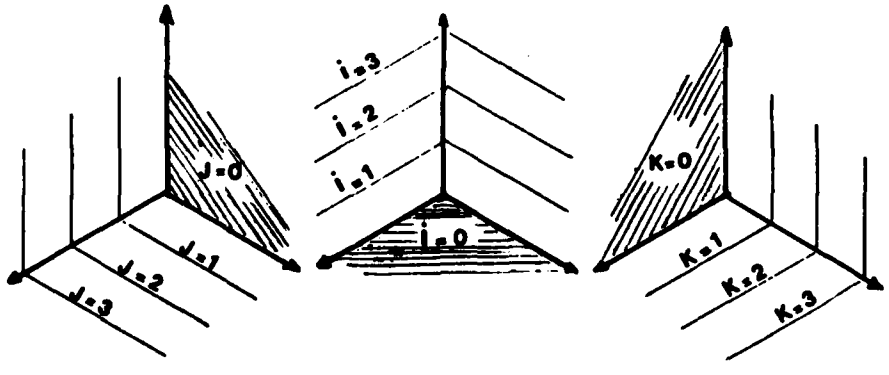
We next apply the model of Section 4 to verify that the network of Figure 5 does indeed produce the elements of the product matrix $C = AB$.

We follow the same scheme used in [5] to label each node in the network by a triplet (i,j,k), $0 \leq i,j,k \leq n$, where i, j and k are the relative position of the node with respect to the three coordinates shown in the figure. Note that we did not label the input cells as these do not affect the verification procedure.

If we assign the colors y, z and x to the edges directed to the south-west, south-east and north, respectively, and we use the convention of Section 4 for labeling the edges and identifying the corresponding sequences, we may describe the operation of each cell by three sequence equations one for each output link, as follows:



(a) A self-timed network for full_matrix multiplication.



(b) The coordinate system used in (a).

Figure 5

For cells (i,o,k), $i,k = 1,\ldots,n$

$$\eta_{i-1,o,k-1} = \Omega_{b_{i-1,k-1}} \eta_{i,o,k}$$

$$\zeta_{i,o,k+1} = \zeta_{i,o,k}$$

$$\xi_{i+1,o,k} = [\xi_{i,o,k} + \eta_{i,o,k} * \zeta_{i,o,k}]$$

For cells (i,j,o), $i,j = 1,\ldots,n$

$$\eta_{i,j+1,o} = \eta_{i,j,o}$$

$$\zeta_{i-1,j-1,o} = \Omega_{a_{j-1,i-1}} \zeta_{i,j,o}$$

$$\xi_{i+1,j,o} = [\xi_{i,j,o} + \eta_{i,j,o} * \zeta_{i,j,o}]$$

For cells (o,j,k), $j,k = 1,\ldots,n$

$$\eta_{o,j+1,k} = \eta_{o,j,k}$$

$$\zeta_{o,j,k+1} = \zeta_{o,j,k}$$

$$\xi_{o,j-1,k-1} = \Omega_o[\xi_{o,j,k} + \eta_{o,j,k} * \zeta_{o,j,k}]$$

For cells (o,o,ℓ), (o,ℓ,o) and (ℓ,o,o), $\ell = 0,\ldots,n$

$$\eta_{i,j+1,k} = \eta_{i,j,k}$$

$$\zeta_{i,j,k+1} = \zeta_{i,j,k}$$

$$\xi_{i+1,j,k} = [\xi_{i,j,k} + \eta_{i,j,k} * \zeta_{i,j,k}]$$

Note that the basic pattern of the three equations is the same for all the cells in the network (all the cells are multiply/add cells). However, different cases have to be considered to account for the initial states of the different links.

The next step isthe specification of the data sequences on the input links of the network. Namely

$$\eta_{n,o,k} = \Omega_{b_{n,k}} \Omega_o^{n-k} \delta^* \qquad k = 0,\ldots,n$$

$$\eta_{i,o,n} = \Omega_{b_{i,n}} \delta^* \qquad i = 1,\ldots,n$$

$$\zeta_{n,j,o} = \Omega_{a_{j,n}} \Omega_o^{n-j} \delta^* \qquad j = 0,\ldots,n$$

$$\zeta_{i,n,o} = \Omega_{a_{n,i}} \delta^* \qquad i = 1,\ldots,n$$

$$\xi_{o,n,k} = \Omega_o \delta^* \qquad k = 0,\ldots,n$$

$$\xi_{o,j,n} = \Omega_o \delta^* \qquad j = 1,\ldots,n$$

where $\delta^*(t)=\delta$ for any t. It is clear from the above equations that no significant (non-δ) input is required besides the initial data. Only some zeroes are required on the links $y_{n,o,k}$ and $z_{n,j,o}$, $j,k = 0, \ldots, n$, to flush the results of the computation. These results would be otherwise destroyed by

the addition of δ quantities to them.

The system of difference equations describing the cells in the network may then be solved, with the given input, to obtain an explicit description of the outputs on the links $x_{n+1,j,o}$ and $x_{n+1,o,k}$, $j,k = 0,\ldots,n$. The solution is rather complex and very similar to the one given in [15] for the band matrix multiplication network. We only give here the results

$$\xi_{n+1,j,o} = \gamma_j \qquad j = 0,\ldots,n$$

$$\xi_{n+1,o,k} = \bar{\gamma}_k \qquad k = 1,\ldots,n$$

where

$$\gamma_j(t) = \begin{cases} \sum_{\ell=0}^{n} a_{j+t-1,\ell}\, b_{\ell,t-1} = c_{j+t-1,t-1} & t \leq n-j+1 \\ \delta & t > n-j+1 \end{cases}$$

and

$$\bar{\gamma}_k(t) = \begin{cases} \sum_{\ell=0}^{n} a_{t-1,\ell}\, b_{\ell,k+t-1} = c_{t-1,k+t-1} & t \leq n-k+1 \\ \delta & t > n-k+1 \end{cases}$$

That is the $(n-j+1)$ elements of the j-th upper diagonal, $0 \leq j \leq n$, of the product matrix C, will appear on the output link $z_{n+1,j,o}$ and the $(n-k+1)$ elements of the k-th lower diagonal, $1 \leq k \leq n$, of C, will appear on $z_{n+1,o,k}$.

6. CONCLUDING REMARKS

For the class of self-timed networks considered in this paper we assumed that each cell does not read its input unless all the input data are available, and does not write its results until all previous results are consumed. Of course, this is not the only organization for self-timed networks. For example, a cell may be allowed to read any input as soon as it is available, compute the results that depend on the read input, and write these results, without waiting for all the other input data to be available. Typically, a cell that copies an input item to an output link may start the copying even if its other inputs are not yet available. Although such organizations were not considered, the techniques presented in this paper may be easily modified and applied to the verification of other types of self-timed systems.

For this class of networks, we proved that the liveness of any computation performed on a given network is uniquely dependent on the initial state. In this context, any simulation may provide a verification of liveness. We also proved that a network is deadlock-free if and only if each cell does complete the execution of its first cycle in any execution history. This means that only finite parts of simulations are needed for the verification of deadlock-free operations. Finally, we derived the sufficient and necessary conditions that may be used to determine the liveness of a network without any simulation. Once a self-timed computation is verified to be deadlock-free, the model of Section 4 may then be used to verify its results.

It was possible to define the general model of Section 4 such that the verification of liveness is implicitly included in the verification of the results. More specifically, data sequences might have been defined as partial functions (rather than total functions) from N^+ to $\bar{R}_\delta$, with the interpretation that the t-th element of a sequence ξ_v associated with a link x_v

is undefined if and only if the destination cell of x_v does not complete the read in its t-th cycle. The relations between the domains of the different sequences should then be defined by additional equations thus complicating the system of equations that models a computation. Although this approach is attractive theoretically, it seemed to be very complex for practical applications, and a separation of the two components of the verification was prefered.

REFERENCES

1) Bentley, J. L. and Kung, H. T., "A Tree Machine for Searching Problems", Proc. of the 1979 International Conference on Parallel Processing (1979), pp. 257-266.

2) Brookes, S., Hoare, C. and Roscoe, A., "A Theory of Communicating Sequential Processes", J. of ACM 31,3 (1984).

3) Caspi, P. and Halbwachs, N., "Algebra of Events: A Model for Parallel and Real Time Systems", Proc. of the 1982 International Conference on Parallel Processing (1982), pp. 150-159.

4) Chandy, K. and Misra, J., "Deadlock Absence Proofs for Networks of Communicating Processes", Information Processing Letters 9,4 (1979), pp. 185-189.

5) Chen, M., "Space-Time Algorithms: Semantics and Methodology", Ph.D. Thesis, California Institute of Technology, (May 1983).

6) Chen. M. and Mead, C., "Formal Specification of Concurrent Systems", USC Workshop on VLSI and Modom Signal Processing, (November 1982).

7) Fisher, A. and Kung, H. T., "Synchronizing Large VLSI Processor Arrays", Proc. of the 10th Annual Symposium on Computer Architecture (1983), pp. 54-58.

8) Kung, H. T., "Why Systolic Architecture", Computer 15 (1982), pp. 37-46.

9) Kung, S. Y., Arun, K., Gal-Ezer, R. and Rao, B., "Wavefront Array Processor: Language, Architecture and Applications", IEEE Trans. on Computers 31,11 (1982), pp. 1054-1066.

10) Kung, S. Y. and Gal-Ezer, R., "Synchronous Versus Asynchronous Computation in VLSI Array Processors", in Real Time Signal Processing V, SPIE Vol. 341 (1982), pp. 53-63.

11) Lamport, L., "Time, Clocks and the Ordering of Events in a Distributed System", CACM 21,7 (1978), pp. 558-565.

12) Leiserson, C., "Systolic Priority Queues", Proc. of Conference on VLSI: Architecture, Design, Fabrication (January 1979), California Institute of Technology.

13) McFarland, M. and Parker, C., "An Abstract Model of Behavior for Hardware Descriptions", IEEE Trans. on Computers 32,7 (1983), pp. 621-637.

14) Melhem, R., "An Abstract Systolic Model and its Application to the Design of a Finite Element System", Ph.D. Thesis (December 1983), Dept. of Computer Science, University of Pittsburgh.

15) Melhem, R. and Rheinboldt, W., "A Mathematical Model for the Verification of Systolic Networks", SIAM J. on Computing 13,3 (1984), pp. 541-565.

16) Mishra, B. and Clarke, E., "Automatic and Hierarchical Verification of Asynchronous Circuits Using Temporal Logic", Tech. Rept. CMU-CS-83 (September 1983), Dept. of Computer Science, Carnegie-Mellon University.

17) Seitz, C., "System Timing", in Introduction to VLSI Systems, ed. by C. Mead and L. Conway, Addison-Wesley (1980), pp. 218-262.

18) Weiser, U. and Davis, A., "A Wavefront Notation Tool for VLSI Array Design", in VLSI Systems and Computations, ed. by H. T. Kung, B. Sproull and G. Steele, Computer Science Press, Rockville, MD (1981), pp. 226-234.

APPENDIX

Proof of Theorem 1: Let $V_1,V_2,\ldots,V_m$ be a finite S^o-feasible sequence such that $S^o \xrightarrow{V_1} S^1 \ldots \xrightarrow{V_m} S^m$ and S^m is a dead state. Let also $W_1,W_2,\ldots$ be any other S^o-feasible sequence such that $S^o \xrightarrow{W_1} \bar{S}^1 \xrightarrow{W_2} \bar{S}^2 \ldots$. We will prove by induction that for any $k \geq 1$ and any $I \in W_k$, there exist a q such that $I \in V_q$. But $V_1,\ldots,V_m$ may contain only a finite number of instances of events, and each may occur only once in a feasible sequence. Hence, $W_1,W_2,\ldots$ contains only a finite number of instances of events and hence is finite. Following is the induction proof.

for $k = 1$: Let $I \in W_1$ and assume that $I \notin V_q$, $q = 1,\ldots,m$. I is feasible at S^o, and hence by Lemma 2 it is feasible at S^m, which contridicts the hypothesis that S^m is a dead state. Hence, $I \in V_q$, for some q.

for any $k > 1$: Assume that *each* instance of event in $W_1,\ldots,W_{k-1}$ does occur in $V_1,\ldots,V_m$, and consider any $I \in W_k$, where I may be a instance of $E_{r,i}$, $E_{c,i}$ or $E_{w,i}$ for some i. Here we will only prove the induction step for $I = E^{(t)}_{r,i}$, for some t. The proof in the other two cases is similar. We first define the sets $R_i = \{(j,i) \mid (j,i) \in \text{dom}(E_{r,i}),\ j \neq i,\ S^o_{j,i} = 1\}$, and $W_i = \{(j,i) \mid (j,i) \in \text{dom}(E_{r,i}),\ S^o_{j,i} = -1\}$. If $t = 1$, then by Lemma 3 $E^{(1)}_{w,j} < E^{(1)}_{r,i}$ for each $(j,i) \in W_i$, that is these $E^{(1)}_{w,j}$ do occur in $W_1,\ldots,W_{k-1}$, and by the induction hypothesis, they also occur in $V_1,\ldots,V_m$. By the state transformation axiom, each $E^{(1)}_{w,j}$ sets the corresponding (j,i)-th entry of the state matrix to 1. Noting that $S^o_{j,i} = 1$ for

$(j,i) \in R_i \cup \{(i,i)\}$ and that an entry in the i-th column of the state matrix that is equal to 1 may change only by the occurence of an instance of $E_{r,i}$, we conclude that $E_{r,i}^{(T)}$, for some $T \geq 1$ does occur in $V_1,\ldots,V_m$, because otherwise $S_{j,i}^m$ would be equal to 1 for any $(j,i) \in dom(E_{r,i})$ which contradicts the hypothesis that S^m is a dead state. Of course if $T > 1$ then $E_{r,i}^{(1)} < E_{r,i}^{(T)}$ and hence $E_{r,i}^{(1)}$ also occurs in $V_1,\ldots,V_m$. Now if $I = E_{r,i}^{(t)}$, $t > 1$, then from Lemma 3, $E_{w,i}^{(t-1)} < E_{r,i}^{(t)}$ and $E_{w,j}^{(t+a_j)} < E_{r,i}^{(t)}$ for each $(j,i) \in R_i \cup W_i$, where $a_j = -1$ and 0 for $(j,i) \in R_i$ and $(j,i) \in W_i$, respectively. That is these $E_{w,i}^{(t-1)}$ and $E_{w,j}^{(t+a_j)}$ do occur in $W_1,\ldots,W_{k-1}$, and by the induction hypothesis, they also occur in $V_1,\ldots,V_m$. By the state transformation axiom, $E_{w,i}^{(t-1)}$ sets the (i,i)-th entry of the state matrix to 1, and each $E_{w,j}^{(t+a_j)}$ sets the corresponding (j,i)-th entry to 1. Noting that an entry in the i-th column of the state matrix that is equal to 1 may change only by the occurence of an instance of $E_{r,i}$, and that by Lemma 3, $E_{r,i}^{(t-1)}$ occurs in $V_1,\ldots,V_m$ before the occurence of $E_{w,i}^{(t-1)}$ or any $E_{w,j}^{(t+a_j)}$, we conclude that $E_{r,i}^{(t)}$ does also occur in $V_1,\ldots,V_m$ because otherwise $S_{j,i}^m$ would be equal to 1 for any $(j,i) \in dom(E_{r,i})$ and hence S^m would not be a dead state. This completes the proof of the first part of the theorem.

The second part of the theorem is a direct consequence of the first part.■

Proof of Theorem 2: Let $S^0 \xrightarrow{V_1} S^1 \xrightarrow{V_2} \ldots$ and let F_i be the set of events feasible at S^i. If $m = 3n$, then the $3n$ instances of events that occured before S^{3n} should be $E^{(1)}_{x,k}$, $x = r,c,w$, $k = 1,\ldots,n$, and it is easy to show that $S^{3n} = S^0$. Now we may construct the infinite S^0-feasible sequence

$$V_1,\ldots,V_{3n}, V^2_1,\ldots,V^2_{3n}, V^3_1,\ldots \text{ with } V^t_i = \{E^{(t)}_{x,k}\} \text{ where } V_i = \{E^{(1)}_{x,k}\}.$$

This proves, by Theorem 1, that S^0 is live. Now let $m \neq 3n$. Clearly m can't be larger than $3n$ because we may have only $3n$ events. Hence $m < 3n$, and there exist a non-empty set of events A such that for each $e \in A$; $e \notin F_m$ and $e^{(1)}$ does not occur in $V_1,\ldots,V_m$. If $V_{m+1} = \{v^{(t)}\}$ for some $v \in F_m$ and $t > 1$ then from Lemma 4 $dom(e) \cap dom(v) = \phi$ for any $e \in A$. But the occurence of $v^{(t)}$ may change only the entires of S^m in $dom(v)$, and hence any $e \in A$ will not be feasible at S^{m+1}. By repeating this argument we may prove that no instances of an event $e \in A$ will occur in $V_1,V_2,\ldots$. Pick any event $E_{x,i} \in A$. From Lemma 3, $E^{(2)}_{r,i}$, $E^{(2)}_{c,i}$ and $E^{(2)}_{w,i}$ do not occur in $V_1,V_2,\ldots$ because this would require the occurence of $E^{(1)}_{r,i}$, $E^{(1)}_{c,i}$ and $E^{(1)}_{w,i}$, thus contradicting that $E_{x,i} \in A$ for some $x = r$, c or w. Next, we will prove that any event $E_{y,j} \notin A$ does occur in $V_1,V_2,\ldots$ only finitely many times. From the connectivity of the network, there is either a path from cell j to cell i, or a path from cell i to cell j. If the first case applies then by repeatedly applying Lemma 3 we may find a T such that $E^{(2)}_{r,i} < E^{(T)}_{w,j}$, and as $E^{(2)}_{r,i}$ does not occur in $V_1,V_2,\ldots$, then $E^{(T)}_{w,j}$, and consequently $E^{(T+1)}_{r,j}$ and $E^{(T+1)}_{c,j}$, do not occur in $V_1,V_2,\ldots$. Similarly, if there is a path from cell i to cell j, then we may find a σ such that $E^{(2)}_{w,i} < E^{(\sigma)}_{r,j}$, and prove that $E^{(\sigma)}_{r,j}$, $E^{(\sigma)}_{w,j}$ and $E^{(\sigma)}_{c,j}$ do not occur in $V_1,V_2,\ldots$. This proves that only finitely many event instances may occur in $V_1,V_2,\ldots$.

Hence, $V_1, V_2, \ldots$ is a finite S^0-feasible sequence and by Theorem 1 S^0 is not a live state. ■

Proof of Theorem 3: Let $G_0 = (V,E,L)$ be an initial state graph of N, and let S^0 be the initial state matrix corresponding to any numbering of the cells of N. First, assume that C1 is violated. Then, there exist a cycle $e_1,\ldots,e_m$ such that $S_{e_i,e_{i+1}} = -1$, $i = 1,\ldots,m-1$, and $S_{e_m,e_1} = -1$. By (3.4c) and (3.4a) of Lemma 3, this gives

$$E^{(1)}_{w,e_1} < E^{(1)}_{r,e_2} < E^{(1)}_{w,e_2} < E^{(1)}_{r,e_3} \ldots < E^{(1)}_{w,e_m} < E^{(1)}_{r,e_1}.$$

But from (3.4a), $E^{(1)}_{r,e_1} < E^{(1)}_{w,e_1}$. Hence, neither $E^{(1)}_{w,e_1}$ nor $E^{(1)}_{r,e_1}$ do occur in any S^0-feasible sequence. As shown in the proof of Theorem 2, this may be used to prove that the initial state is not live. The same technique may be applied to prove that the initial state is not live if any of C2 or C3 is violated.

Now assume that all of C1, C2 and C3 are satisfied in G_0. If the algorithm LIVE is applied to S^0, then during its execution, we may keep a partitioning of the nodes in G_0 into three sets T_0, T_r and T_w as follows:

- Initially, set $T_0 = V$, $T_r = T_w = \phi$.
- Whenever an instance $E^{(1)}_{r,i}$ is simulated in LIVE, move node i from T_0 to T_r.
- Whenever an instance $E^{(1)}_{w,i}$ is simulated in LIVE, move node i from T_r to T_w.

LIVE may be blocked only if every column i with Mark[i] = false contains at least a -1 entry, and every row j with Mark[j] = false contains at least a 1 entry. Noting that 1) Mark[e] = True for any $e \in T_w$, 2) if $e \in T_r$ and $v \in T_0$ then $S[e,v] = S^0_{e,v}$, and 3) if $e \in T_r$ and $v \in T_r \cup T_w$ then

$S[e,v] \neq 1$, it may be easily shown that if LIVE is blocked, then

B1) For any node $e \in T_0$, there exist at least one IWE link directed from a node $v \in T_0 \cup T_r$ to e.

B2) For any node $v \in T_r$, there exist at least one IRE link directed to v from a node $e \in T_0$.

In order to prove that the initial state is live, we will prove, by contradiction, that if C1, C2 and C3 are satisfied in G_0, then there cannot exist two disjoin subsets $T_0, T_r \subseteq G_0$ such that both B1 and B2 are satisfied.

Assume that such T_0 and T_r do exist. We will first prove that any node $e_0 \in T_0$ is W-reachable from a node $v_0 \in T_r$. Consider any $e_0 \in T_0$. By B1, there exist $e_1 \in T_0 \cup T_r$ such that $e_1 \rightsquigarrow e_0$. If $e_1 \in T_r$ then $v_0 = e_1$ and we are done, else $e_1 \in T_0$. By B1, there exist $e_2 \in T_0 \cup T_r$ such that $e_2 \rightsquigarrow e_1$. If $e_2 \in T_r$ then $v_0 = e_2$ and we are done, else $e_2 \in T_0$ and by C1 $e_2 \neq e_i$, $0 \leq i < 2$. This argument may be repeated at most d times, where d is the number of nodes in T_0, before a node $v_0 \in T_r$ with $v_0 \rightsquigarrow e_0$ is found.

Next, we will construct two subsets $A_0 \subseteq T_0$ and $A_r \subseteq T_r$ as follows:

1) Start with $A_0 = \{e_0\}$ where e_0 is any node in T_0 and $A_r = \{v_0\}$ where $v_0 \in T_r$ and $v_0 \rightsquigarrow e_0$.

2) For $k = 1,2,\ldots$

   2.1) Chose $e_k \in T_0 \sim A_0$ such that $v_{k-1} \rightarrow e_k$. The existence of such e_k is discussed later. Set $A_0 = A_0 \cup \{e_k\}$.

   2.2) Chose $v_k \in T_r \sim A_r$ such that $v_k \rightsquigarrow e_k$. Set $A_r = A_r \cup \{v_k\}$.

The existence of $e_k \in T_o$, $v_{k-1} \rightarrow e_k$ is guaranteed by B2. Also $e_k \notin A_o$ because if $e_k = e_{k-1}$ then by construction $v_{k-1} \rightsquigarrow e_{k-1}$ and $v_{k-1} \rightarrow e_{k-1}$, which contradicts C2, and if $e_k = e_j$ for some $j < k-1$, then $v_{k-1} \rightarrow e_j$, $v_{i-1} \rightarrow e_i$, $i = j+1,\ldots,k-1$ and $v_i \rightsquigarrow e_i$, $i = j,\ldots,k-1$. This clearly contradicts C3. The existence of v_k is step 2.2 follows from a similar argument.

However, T_o and T_r are finite and hence there will exist a step k where either $e_k \in A_o$ or $v_k \in A_r$, which contradicts one of C2 or C3, as shown previously. Hence if C1, C2 and C3 are satisfied in G_o, then a partitioning of G_o that satisfies B1 and B2 cannot exist. ■